Date=27/10/2020
Lecture By=Manish Mahant
Subject ⇒ Functions and Jquery Revision

IN PREVIOUS LECTURE **(QUICK RECAP) Date-26/10/2020**	**In Today's Lecture (Overview)**
The HTML DOM (Document Object Model) What is the DOM? JavaScript - HTML DOM Methods HTML DOM appendChild() Method Window setTimeout() Method Window setInterval() Method Questions for self practice / Assignment/CC For The Day	What is functions Defining functions Calling functions Function scope What is Jquery Why jQuery? jQuery Syntax Questions For Self Practice // CC For the Day

What is functions

Functions are one of the fundamental building blocks in JavaScript. A function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output. To use a function, you must define it somewhere in the scope from which you wish to call it.

Defining functions

A **function definition** (also called a **function declaration**, or **function statement**) consists of the `function` keyword, followed by:

- The name of the function.
- A list of parameters to the function, enclosed in parentheses and separated by commas.

- The JavaScript statements that define the function, enclosed in curly brackets, `{ ... }`

```
function square(number) {
    return number * number;
  }
```

The function `square` takes one parameter, called `number`. The function consists of one statement that says to return the parameter of the function (that is, `number`) multiplied by itself. The statement `return` specifies the value returned by the function:

```
return number * number;
```

If you pass an object (i.e. a non-primitive value, such as `Array` or a user-defined object) as a parameter and the function changes the object's properties, that change is visible outside the function, as shown in the following example:

```
function myFunc(theObject) {
    theObject.make = 'Toyota';
  }

  var mycar = {make: 'Honda', model: 'Accord', year: 1998};
  var x, y;

  x = mycar.make; // x gets the value "Honda"

  myFunc(mycar);
  y = mycar.make; // y gets the value "Toyota"
                  // (the make property was changed by the function)
```

While the function declaration above is syntactically a statement, functions can also be created by a function expression.

Such a function can be **anonymous**; it does not have to have a name. For example, the function `square` could have been defined as:

```
const square = function(number) { return number * number } var x = square(4) // x gets the value 16
```

In the following code, the function receives a function defined by a function expression and executes it for every element of the array received as a second argument.

```
function map(f, a) {
    let result = []; // Create a new Array
    let i; // Declare variable
    for (i = 0; i != a.length; i++)
      result[i] = f(a[i]);
    return result;
  }
  const f = function(x) {
     return x * x * x;
  }
  let numbers = [0, 1, 2, 5, 10];
  let cube = map(f,numbers);
  console.log(cube);
```

Calling functions

Defining a function does not *execute* it. Defining it simply names the function and specifies what to do when the function is called.

Calling the function actually performs the specified actions with the indicated parameters. For example, if you define the function `square`, you could call it as follows:

```
square(5);
```

The preceding statement calls the function with an argument of `5`. The function executes its statements and returns the value `25`.

Functions must be *in scope* when they are called, but the function declaration can be hoisted (appear below the call in the code), as in this example:

```
console.log(square(5));
/* ... */
function square(n) { return n * n }
```

Function scope

Variables defined inside a function cannot be accessed from anywhere outside the function, because the variable is defined only in the scope of the function. However, a function can access all variables and functions defined inside the scope in which it is defined.

In other words, a function defined in the global scope can access all variables defined in the global scope. A function defined inside another function can also access all variables defined in its parent function, and any other variables to which the parent function has access.

```
// The following variables are defined in the global scope
var num1 = 20,
    num2 = 3,
    name = 'Chamahk';

// This function is defined in the global scope
function multiply() {
  return num1 * num2;
}

multiply(); // Returns 60

// A nested function example
function getScore() {
  var num1 = 2,
      num2 = 3;

  function add() {
    return name + ' scored ' + (num1 + num2);
```

```
  }

  return add();
}

getScore(); // Returns "Chamahk scored 5"
```

You may nest a function within another function. The nested (inner) function is private to its containing (outer) function.

It also forms a *closure*. A closure is an expression (most commonly, a function) that can have free variables together with an environment that binds those variables (that "closes" the expression).

Since a nested function is a closure, this means that a nested function can "inherit" the arguments and variables of its containing function. In other words, the inner function contains the scope of the outer function.

To summarize:

- The inner function can be accessed only from statements in the outer function.
- The inner function forms a closure: the inner function can use the arguments and variables of the outer function, while the outer function cannot use the arguments and variables of the inner function.

The following example shows nested functions:

```
function addSquares(a, b) {
    function square(x) {
      return x * x;
    }
    return square(a) + square(b);
  }
  a = addSquares(2, 3); // returns 13
  b = addSquares(3, 4); // returns 25
  c = addSquares(4, 5); // returns 41
```

Since the inner function forms a closure, you can call the outer function and specify arguments for both the outer and inner function:

```
function outside(x) {
    function inside(y) {
      return x + y;
    }
    return inside;
  }
  fn_inside = outside(3); // Think of it like: give me a function that
adds 3 to whatever you give
                          // it
  result = fn_inside(5); // returns 8

  result1 = outside(3)(5); // returns 8
```

What is Jquery

jQuery is a lightweight, "write less, do more", JavaScript library. The purpose of jQuery is to make it much easier to use JavaScript on your website. jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

Tip: In addition, jQuery has plugins for almost any task out there.

Why jQuery?

There are lots of other JavaScript libraries out there, but jQuery is probably the most popular, and also the most extendable.

Many of the biggest companies on the Web use jQuery, such as:

- Google
- Microsoft
- IBM
- Netflix

jQuery Syntax

The jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s).

Basic syntax is: $(*selector*).*action*()

- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

Examples:

`$(this).hide()` - hides the current element.

`$("p").hide()` - hides all <p> elements.

`$(".test").hide()` - hides all elements with class="test".

`$("#test").hide()` - hides the element with id="test".

To know more about Jquery “Click Here”

Questions For Self Practice // CC For the Day

https://au-assignment.s3.ap-south-1.amazonaws.com/Week_18_Day_2_Challenge-bb2ed5f4-ac0a-4dd5-bff1-645bf94c1e57.pdf